A Knowledge-based Approach for Automating a Design Method for Concurrent and Real-Time Systems

Kevin L. Mills, National Institute of Standards and Technology
Hassan Gomaa, George Mason University

This paper describes a knowledge-based approach to automate CODARTS, a software design method for concurrent and real-time systems. The approach uses multiple paradigms to represent knowledge embedded within CODARTS. Semantic data modeling provides the means to model specifications and related designs. A specification meta-model enables automated inferences about the presence of semantic concepts within a specification, while a design meta-model permits automated reasoning concerning concurrent designs. Production rules form the basis for modeling a set of heuristics that can generate concurrent designs based upon semantic concepts from the specification and design meta-models. Together, the semantic data models and production rules, encoded using an expert-system shell, compose CODA, an automated designer's assistant. CODA is applied to generate ten concurrent designs for four real-time problems.

1.0 Introduction

Software engineering researchers and practitioners strive to improve the quality of software products by increasing the discipline used during software development. One means of increasing discipline entails the development and application of software design methods. Some researchers attempt to enhance the utility of software design methods by providing automated support. To date, such attempts rely upon either of two approaches: clustering algorithms or rule-based expert systems. Richer knowledge engineering models, integrating semantic data modeling with production rules, can potentially lead to more effective automation of software design methods. This paper describes one approach to automating a software design method by using multiple paradigms to represent the knowledge embedded in the method.

After discussing some existing approaches to automate software design methods, this paper describes CODARTS (COncurrent Design Approach for Real-Time Systems), a software design method for concurrent and real-time systems [1], and then proposes a knowledge-based approach to automate CODARTS. The proposed approach leads directly to CODA (COncurrent Designers Assistant), an automated designer's assistant. The paper describes the use of CODA to generate ten concurrent designs for four real-time problems, and then compares CODA to other approaches that automate software design methods.

2.0 Automating Software Design Methods

A software design method provides a methodical, consistent, and teachable approach that defines what decisions a designer needs to make, when to make them, and, importantly, when to stop making decisions. [2] In addition, a software design method provides a consistent notation that can improve communication among those who must review and understand the meaning of a design. In effect, a software design method encodes knowledge about good design practices into a form that designers can use to manually construct software designs.

Providing automated support for software design methods can lead to several benefits. First, automation can improve the rigor with which a software design method is applied. Automation can ensure that a designer does not overlook any of the myriad details associated with the design process. Automation can establish that constraints levied on a design are satisfied, or that any unsatisfied constraints are brought to the designer's attention. Second, automation can improve a designer's ability to generate alternate designs. Since automation can speed up the generation of designs without sacrificing rigor, a designer can more readily produce several designs from one specification. Third, automation can reduce the variability among the types of designs generated by various designers. Reduced variability of form can increase the ability of customers, analysts, and programmers to understand designs. Fourth, automation can improve the performance of inexperienced designers both immediately, by making default decisions, and gradually, by explaining default design decisions to the designer.

A number of researchers propose approaches to automate software design methods. Four such approaches are described in this section. Three of the four approaches produce a sequential design, represented as structure charts, from a specification represented by data flow diagrams (DFDs). One such approach, CAPO (Computer-Aided Process Organization) [3], relies upon various clustering algorithms to group transformations from a DFD into modules based upon numeric values and weights assigned to represent the amount and frequency of data flow between the

transformations. The second approach, STES (Specification-Transformation Expert System) [4], produces structure charts from a DFD by representing the Structured Design method of Yourdon and Constantine [5] as a set of expert-system rules, encoded using an expert-system shell. The third approach [6] uses an *entity aggregate relationship attribute* model to formally describe a DFD and a structure chart, and then defines transformation rules, based upon set theory, to convert a formal description of a DFD into a formal description of a structure chart. As with STES, the transformation rules implement the Structured Design method.

A fourth approach automates the generation of concurrent designs for SARA (the System ARchitects Apprentice). [7] The SARA Design Assistant encompasses expert-system rules that examine two separate specifications: system verification diagrams, or SVDs, and a related DFD. Each SVD can use constructs representing sequence, three forms of selection (exclusive-or, sequential-inclusive-or, and sequential-exlcusive-or), and parallelism to restrict the concurrency possible among the transformations on a related DFD. The expert-system rules defined for the SARA Design Assistant map elements from a DFD onto constructs contained within the SARA design simulator.

3.0 An Overview Of CODARTS

While the SARA Design Assistant constructs a concurrent design from two separate, but related, specifications, CODARTS [1] uses criteria for information hiding and task structuring to form a concurrent design, including both tasks and information hiding modules [8], from a single specification. CODARTS begins by using COBRA (Concurrent Object-Based Real-time Analysis) to analyze and model a system under design. COBRA, while using RTSA (Real-Time Structured Analysis) notation, provides an alternative to the RTSA [9,10] decomposition strategy that includes guidelines for developing an environmental model based on the system context diagram, and defines structuring criteria for decomposing a system into subsystems, and for determining objects and functions within each subsystem. Finally, COBRA includes a behavioral approach, based on event sequencing scenarios, for determining how the objects and functions within a subsystem interact. A COBRA specification is documented as a hierarchical data/control flow diagram (D/CFD), with a state-transition diagram for each control transformation and a mini-specification for each data transformation, and a data dictionary.

Once a COBRA specification exists, CODARTS provides four steps for generating a concurrent design: 1) Task Structuring, 2) Task Interface Definition, 3) Module Structuring, and 4) Task and Module Integration. First, CODARTS task structuring criteria assist a designer in examining a COBRA specification to identify concurrent tasks. The task structuring criteria, consisting of a set of heuristics derived from experience obtained in the design of concurrent systems, can be grouped into four categories: input/output task structuring criteria, internal task structuring criteria, task cohesion criteria, and task priority criteria. In a given design, a task may exhibit several criteria and many tasks may exhibit the same criteria.

The input/output and internal task structuring criteria identify tasks based upon how and when a task is activated: periodically based on the need to poll a device or to perform a calculation; asynchronously based on an external device interrupt or on an internal event. The task cohesion criteria help a designer to identify COBRA objects and functions that can be combined together within the same task, e.g., because a set of operations: must be performed sequentially (sequential cohesion); can be performed with the same period or with a harmonic period (temporal cohesion); or perform a set of closely related functions (functional cohesion). The task priority criteria prevent a designer from combining tasks that might need to execute at substantially differing priorities.

As a second step, CODARTS provides guidelines for defining inter-task interfaces. Once tasks are defined, data and event flows from a COBRA specification can be mapped to inter-task signals or to tightly-coupled or loosely-coupled messages, depending on the synchronization requirements between specific pairs of tasks.

As a third step, CODARTS includes criteria, based on information hiding, to help a designer identify modules from the objects and functions in a COBRA specification. In general, the CODARTS module structuring criteria form modules to hide the details of: device characteristics, data structures, state-transition diagrams, and algorithms.

As a fourth step, once both the task and module views of a concurrent design exist, CODARTS provides guidelines to help a designer relate the independent views into a single, consistent design. Each task represents a separate thread of control, activated by some event: either an interrupt, a timer, an internal signal, or a message arrival. Each module provides operations that can be accessed by the tasks in a design. CODARTS helps a designer to establish the control flow from events to tasks and then on to operations within modules.

4.0 A Knowledge-based Approach To Automate CODARTS

The CODARTS design method, including COBRA, consists of design knowledge that can be encoded to form the basis for an automated designer's assistant. A separate reference provides a complete description of a knowledge-based approach to automate CODARTS. [11] An overview of the approach appears in the following subsections.

4.1 Conceptual View. Figure 1 presents a conceptual view of the proposed approach. The major components include meta-knowledge bases, design knowledge bases, meta-models, and support knowledge bases. The *meta-knowledge bases* control the design process and interactions with the designer. The *design process meta-knowledge* enforces ordering constraints among the steps within the CODARTS method. The *user-interface meta-knowledge* provides the designer a set of commands to access various design operations. The *design knowledge bases* encode the means to analyze specifications (*specification analysis, inference, and elicitation knowledge*) and to generate designs (*design generation knowledge*). Together, the three *meta-models* define the semantic concepts and semantic relationships that can be reasoned about when generating a design. In essence, the *specification meta-model* enables the modeling of models of real-time problems, the *design meta-model* allows the modeling of models of concurrent designs, and the *target environment description meta-model* allows salient characteristics of target environments to be described The *support knowledge bases* encode specific commands for allowing a designer to query instances of a specification, design, or target environment description, or to describe a new target environment. The remaining three components, shown in Figure 1 as data repositories, represent libraries containing instances of specifications, designs, and target environment descriptions. The three most interesting components, shaded in Figure 1, bear further elaboration.

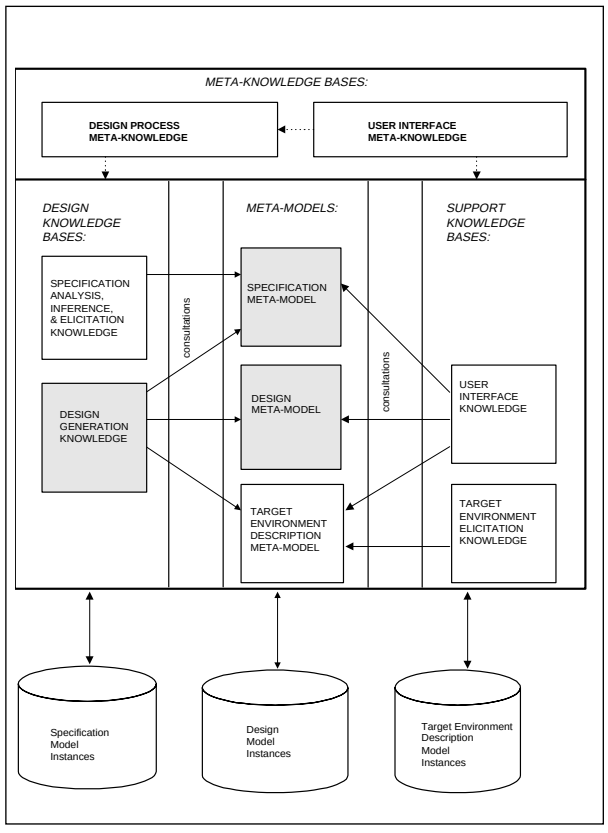


Figure 1. Conceptual View Of Approach

4.2 Specification Meta-Model. The specification meta-model represents the symbols from RTSA and the semantic concepts from COBRA using: 1) a concept hierarchy, where each concept embodies an optional set of axioms, 2) concept classification rules, deployed in an inference network, and 3) logic to elicit missing information. The concept hierarchy establishes **is-a** relationships among the various semantic concepts that can be represented using RTSA and COBRA. Figure 2 depicts a slice through the concept hierarchy from the most general concept, Specification Element, to a leaf concept, Periodic Device Input Object.

Within the concept hierarchy, each concept can be required to satisfy axioms; each concept must also satisfy all axioms along all **is-a** paths leading to that concept. Figure 2 shows concept inheritance with a directed arc pointing from a child concept to its parent concept(s). For each concept, Figure 2 shows the names of the associated axiom(s) in a rectangle connected to the concept using a bi-directional arrow. Dashed, directed arcs connect the axiom rectangles to illustrate that axioms are inherited in accordance with the concept hierarchy. The concept Periodic Device Input Object, even though it contains no axioms of its own, must satisfy twelve axioms based on inheritance. Similar groups of axioms can be composed for any concept in the specification meta-model. The group of axioms that apply to a given concept in the specification meta-model comprise a formal definition for instances of the given concept; thus, any instance that satisfies the applicable axioms for that concept is a valid instance of the concept.

Given a specification expressed using only the symbols of RTSA, the axioms provide the basis for a set of rules that allow semantic concepts from COBRA to be classified within the specification meta-model. Figure 3 illustrates a four-stage inference network composed of concept-classification rules. Stage one of the network, the Arc Classification stage, attempts to fully classify all arcs from the specification, also classifying Terminators and Transformations to the extent necessary to classify the arcs. Stage two of the network, Transformation Classification, attempts to fully classify all transformations from the specification. The Transformation Classification stage accepts the partial classifications from stage one and produces thirteen additional full classifications, as listed in the figure, and two partial classifications. Stage three, Stimulus-Response Classification, resolves the classification of each data flow from the specification that could not be classified during Arc Classification. Stage four, Ambiguous-Function Classification, yields the remaining full classifications. Where the classification rules cannot

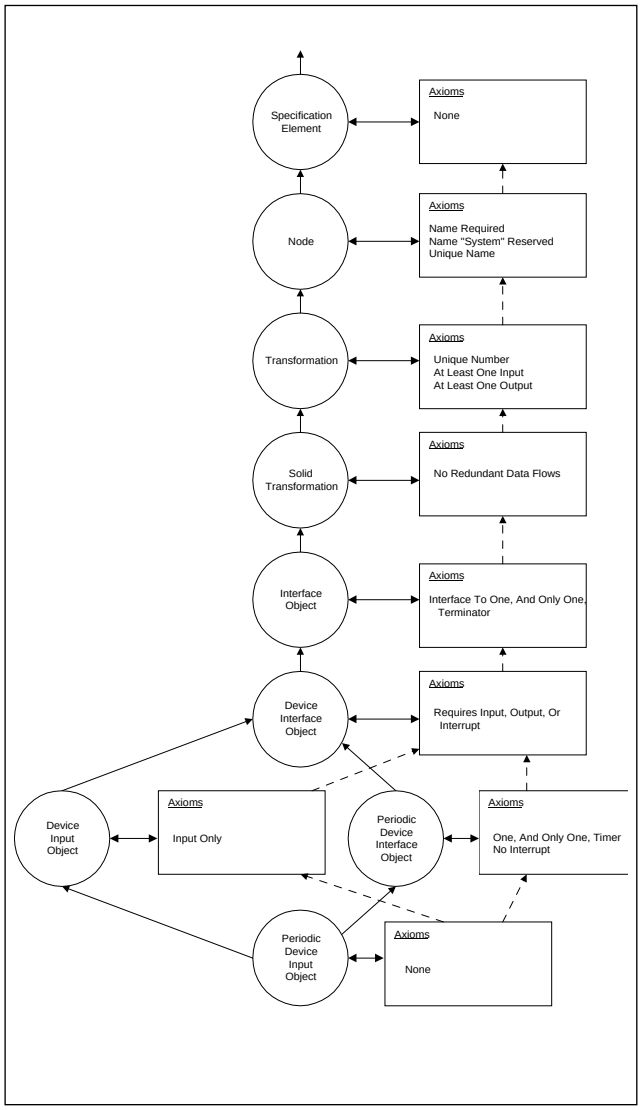


Figure 2. Concept Hierarchy, Axioms, and Inheritance

distinguish among concepts a human analyst is consulted. Figure 3 lists all leaf-level concepts from the specification meta-model; Mills provides a more complete treatment. [11]

4.3 Design Meta-Model. In addition to reasoning about specifications, a designer must also reason about the state of the evolving, concurrent design. The design meta-model allows an automated tool to simulate a designer's semantic view of concurrent designs. The design meta-model consists of 1) an E-R Model of design entities and relationships, 2) constraints on traceability between elements of a specification and elements of a design, 3) constraints among relationships within the design, 4) constraints about target environments, and 5) design guidelines.

Figure 4 illustrates the entities and associated attributes that compose the design meta-model, shows the inheritance relationships among those entities, and also depicts two key relationships in which all design entities participate. Every entity within the design meta-model is a Design Element that can Trace To/From one or more Specification Elements from the specification meta-model; however, certain constraints restrict this relationship to only those that make sense. For example, a Transformation on the specification can lead to a Task or an IHM (information hiding module) but not to a Message. The E-R diagram does not depict these specific constraints. Every Design Element can also Track each Decision made about it; thus, an automated tool can capture design rationale. The remainder of the entities in Figure 4 define the semantic elements used to describe concurrent designs.

Aside from the inheritance relationships and the relationships named Tracks and Traces To/From depicted in Figure 4, the design meta-model includes a number of additional relationships, as shown in Figure 5. These additional relationships increase the richness and complexity of the design meta-model. Each relationship in Figure 5 should be understood to be bi-directional, including both the relationship as shown and its inverse. For the most part, the relationships shown in Figure 5 can be read intuitively. For example, a Task can Read and Write Data, can Generate and Accept an Event, can Send and Receive a Message, can Invoke an Operation, and so on. While Figure 5 depicts cardinality constraints, more complex constraints do not appear on the E-R diagram. For example, each Message entity in a design, where that Message entity is not carried within another Message entity, must be sent and received by one, and only one, task. Complex constraints such as these are represented as query specifications that must hold for valid instances of the design meta-model. Constraints on target environments, such as whether priority message queues are supported, and design guidelines, such as the threshold for task inversion, can be represented independently from the constraints on instances of the design meta-model.

4.4 Design-Generation Knowledge. Using the semantic concepts represented in the specification and design meta-models, a designer can apply various heuristics from the CODARTS design method to produce a concurrent design from a COBRA specification. To facilitate automation of the designer's decision-making, CODARTS heuristics can be encoded into a knowledge-base that includes a partition for each of the four CODARTS design steps (see section 3): Task Structuring, Task Interface Definition, Module Structuring, and Task and Module Integration. Each knowledge partition consists of a sequence of decision-making processes, where each process embodies a set of design rules that encode relevant CODARTS heuristics.

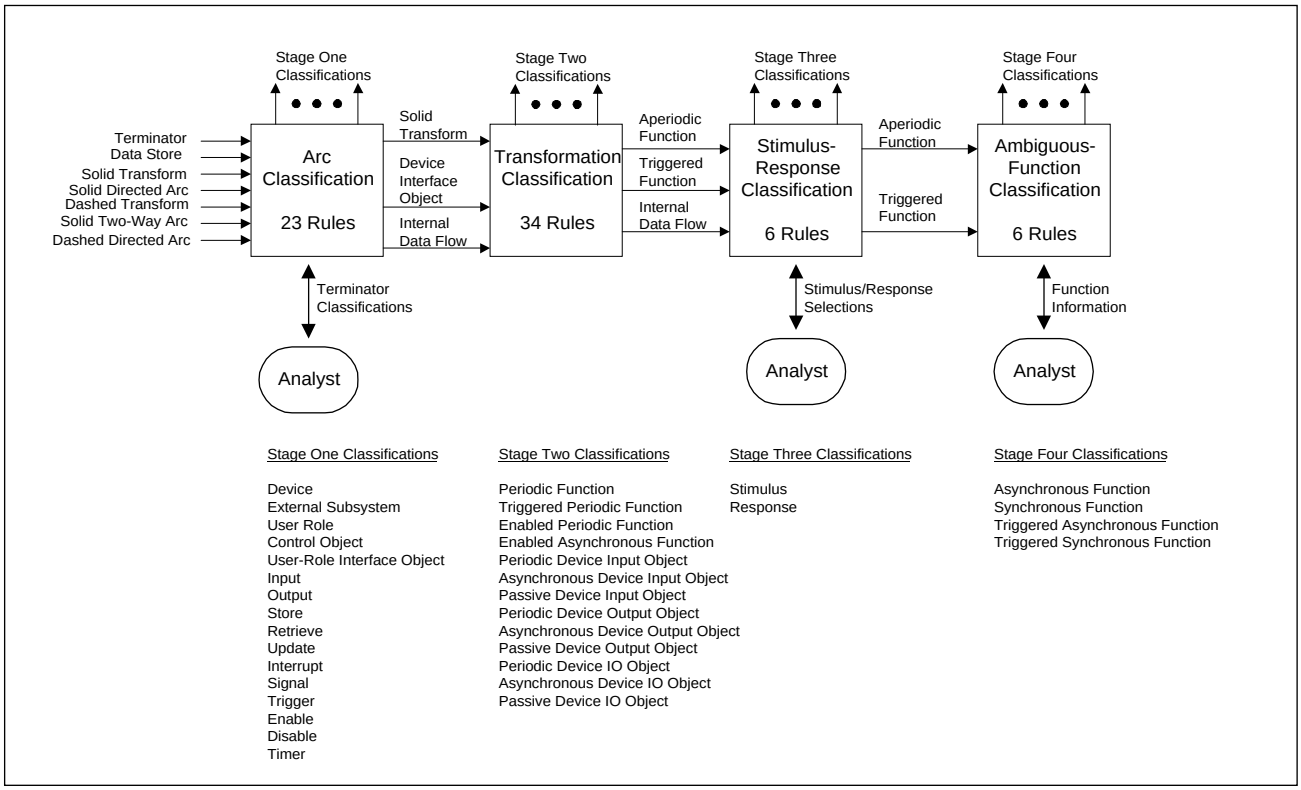


Figure 3. Concept-Classification Inference Network

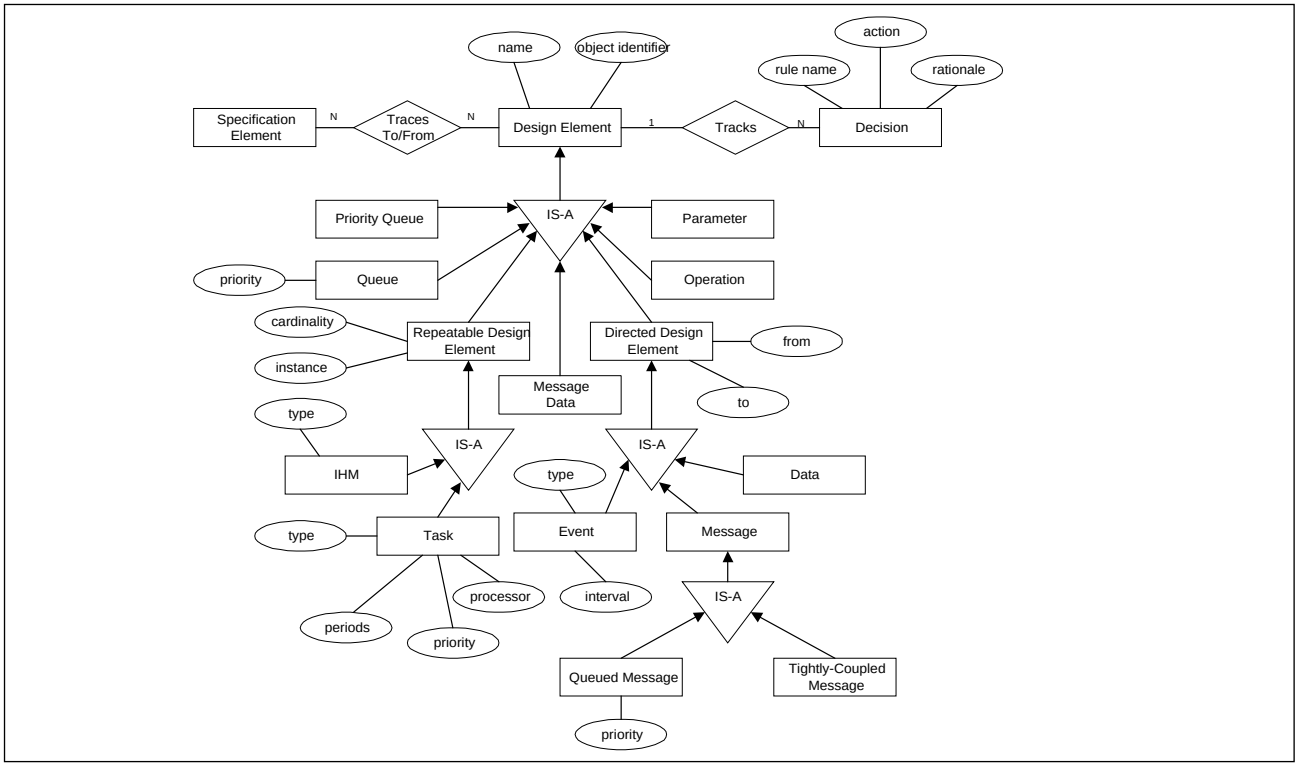


Figure 4. E-R Model Of Design Entities Within The Design Meta-Model

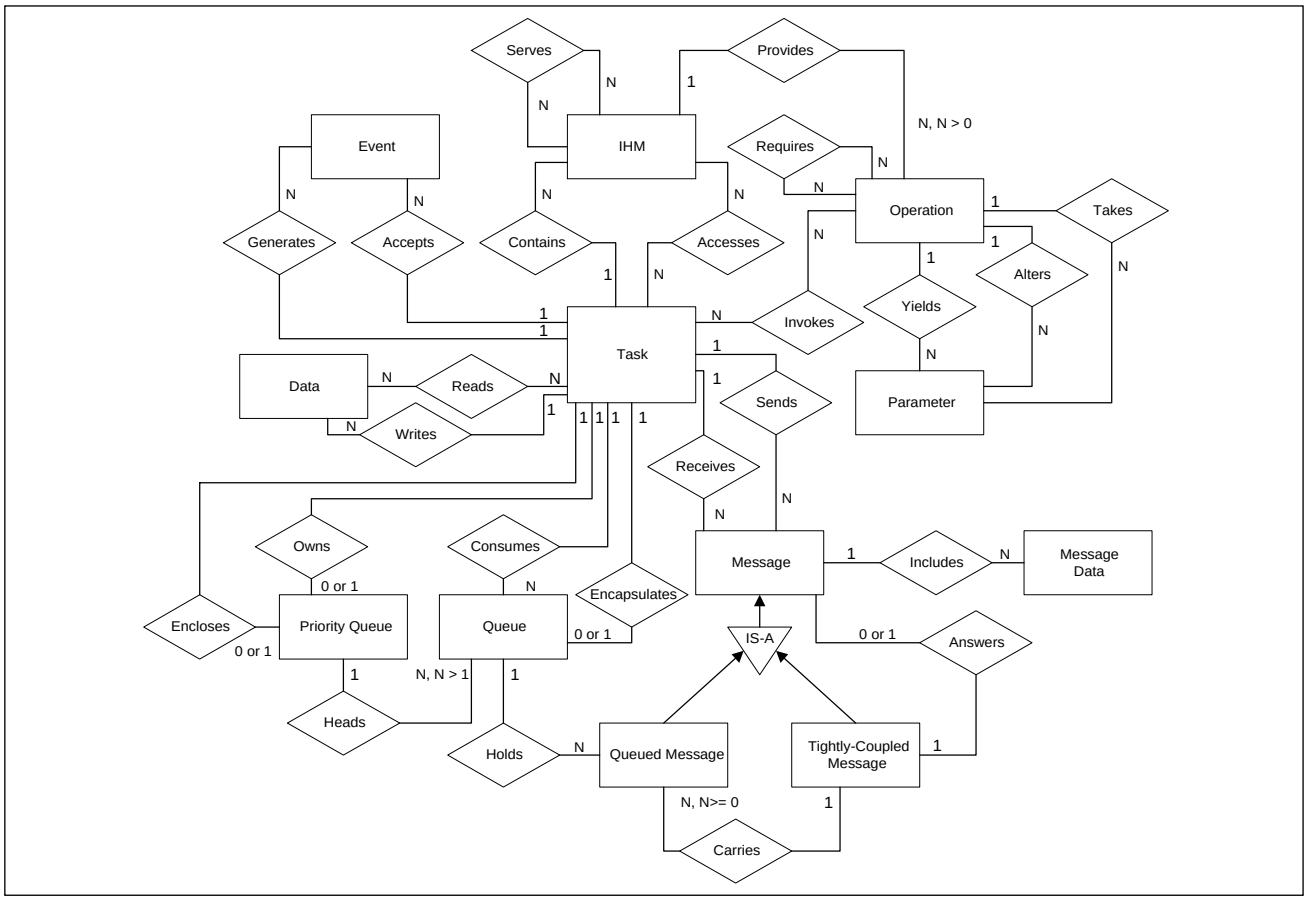


Figure 5. E-R Model Of Design Relationships Within The Design Meta-Model

To illustrate the form and effect of the design rules, some examples might prove useful. Figure 6(a) shows a fragment from a COBRA specification for an automobile cruise control and monitoring system. (A complete treatment of the entire specification appears elsewhere. [11]) Figure 6(c) gives an example design rule that creates a task based upon the CODARTS criterion for identifying periodic device input/output tasks. When applied to the COBRA fragment in Figure 6(a) the rule in Figure 6(c) produces the CODARTS design fragment shown in Figure 6(b). Due to previous execution of the classification rules (see Figure 3), the rule shown in Figure 6(c) recognizes the Brake transformation as an instance of a Periodic Device Interface Object, specifically a Periodic Device Input Object.

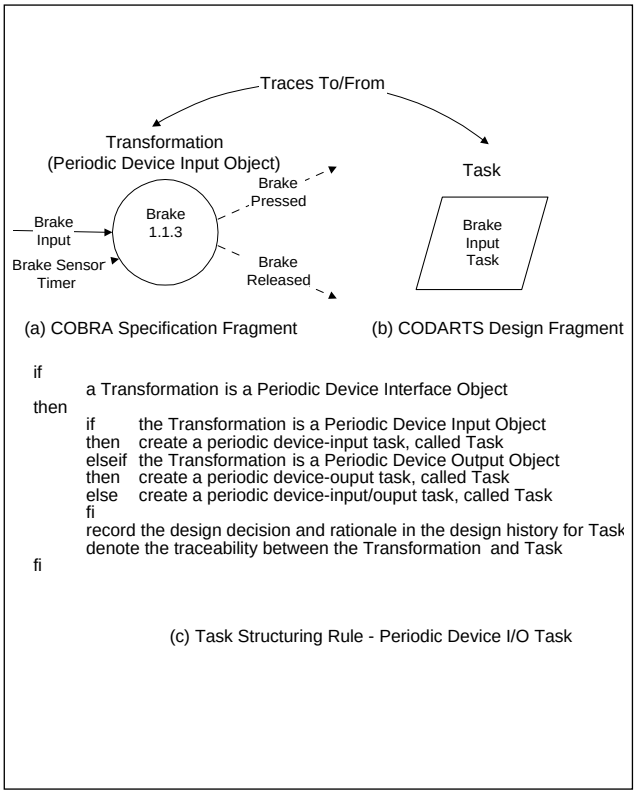
Figure 7(a) shows another fragment from the same COBRA specification. Based on a CODARTS criterion for identifying data-abstraction modules, the design rule in Figure 7(c) creates the CODARTS design fragment shown as Figure 7(b).

Figure 8 illustrates a design rule that examines both the specification and the evolving design. Figure 8(a) shows a specification fragment representing an Interface Object that receives an Interrupt. First, the design rule in Figure 8(c) recognizes that an existing task, shown in Figure 8(b), Traces From the Interface Object in Figure 8(a). Subsequently, the design rule allocates an External Event for the existing task, based on the Interrupt in 8(a).

Figure 9 illustrates a rule that logically places a state-transition module within a particular task. Such logical placement denotes that the state-transition module is executed only within the thread of control of the containing task. Shared modules are placed logically outside of any particular task. The rule shown in 9(c) recognizes cases, such as the specification and design fragments given in 9(a) and 9(b), respectively, where a task and state-transition module both result from the same Control Object; and, thus, the state-transition module is invoked only by the identified task.

5.0 An Automated Designer's Assistant

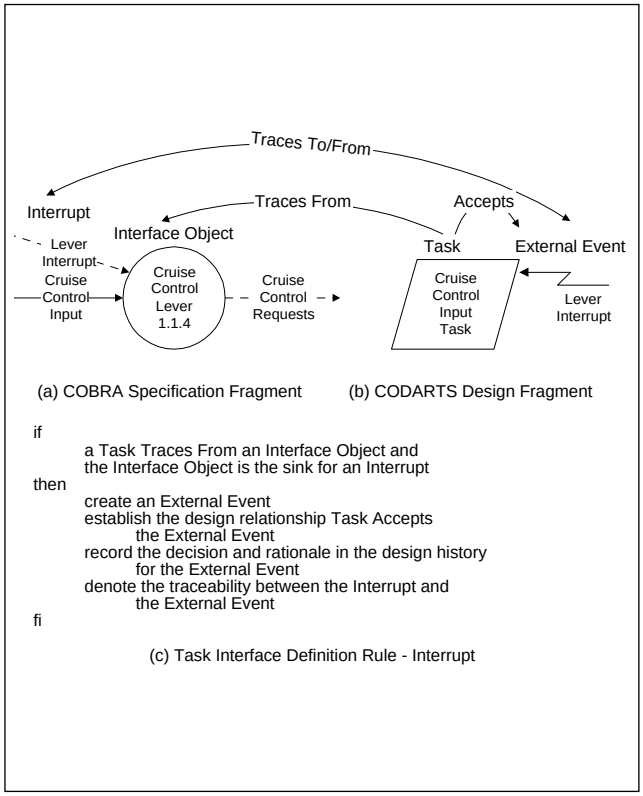
The knowledge described in the preceding sections was mapped onto various knowledge representation techniques provided by an expert-system shell, CLIPS Version 6.0 [12], to produce a COncurrent Designer's Assistant (CODA), which was then applied to four real-time problems that often appear in the literature: an automobile cruise control and




```
if
        a Transformation is a Periodic Device Interface Object
then
        if      the Transformation is a Periodic Device Input Object
        then    create a periodic device-input task, called Task
        elseif  the Transformation is a Periodic Device Output Object
        then    create a periodic device-ouput task, called Task
        else    create a periodic device-input/ouput task, called Task
        fi
        record the design decision and rationale in the design history for Task
        denote the traceability between the Transformation  and Task
fi
```

(c) Task Structuring Rule - Periodic Device I/O Task

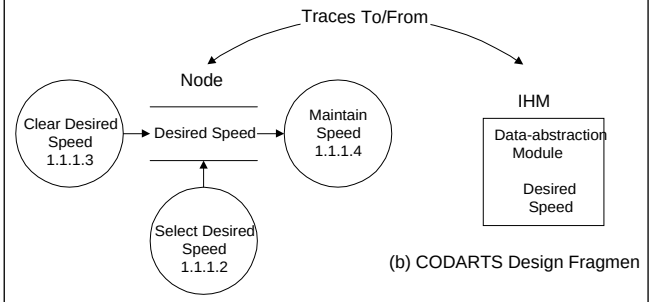
Figure 6. An Example Task Structuring Rule




```
if
        a Task Traces From an Interface Object and
        the Interface Object is the sink for an Interrupt
then
        create an External Event
        establish the design relationship Task Accepts
                the External Event
        record the decision and rationale in the design history
                for the External Event
        denote the traceability between the Interrupt and
                the External Event
fi
```

(c) Task Interface Definition Rule - Interrupt

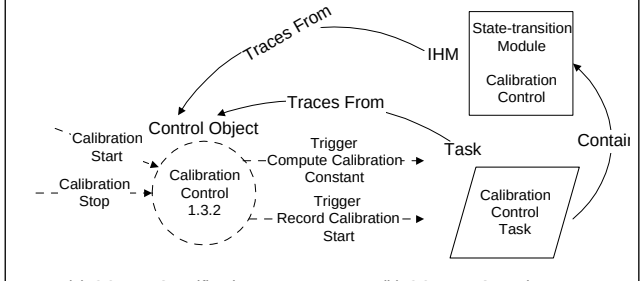
Figure 8. An Example Task Interface Definition Rule




```
if      a Node is a Data Store and
        the Node is accessed by multiple transformations
then
        create a Data-abstraction Module, called IHM
        record the decision and rationale in the design history for IHM
        denote the traceability between the Node and IHM
fi
```

(c) Module Structuring Rule - Data-abstraction Module

Figure 7. An Example Module Structuring Rule




```
if
        an IHM is a state-transition module and
        the IHM Traces From a Control Object and
        a Task Traces From the same Control Object
then
        establish the design relationship that the
                Task Contains the IHM
        record the decision and rationale in the design
                history for the IHM
fi
```

(c) Task And Module Integration Rule - Place State-transition Module Within Tas

Figure 9. An Example Task And Module Integration Rule

monitoring system, a robot controller, an elevator control system, and a remote temperature sensor [11]. For each problem, CODA analyzed a specification and then generated one or more concurrent designs.

During analyses, CODA classified, automatically, 81% of the specification elements, while 5%, were data stores, which are directly representable using RTSA notation. The remaining 14% were classified after interaction between CODA and a designer In 8% of the cases CODA asked the designer whether a terminator represented a device, external subsystem, or user role. The remaining classification decisions where CODA required help split between two categories: 2% where CODA made a tentative classification that the designer had to confirm or override and 4% where CODA required additional information from the designer in order to make a classification.

During design generation, CODA made 97%, of the decisions without consulting a designer. In the instances where CODA did consult a designer, half involved decisions about synchronization requirements surrounding messages exchanged between tasks, 36% involved decisions about how to allocate a node from a specification among two or more tasks or modules, and the rest required judgments about design optimizations.. Where an inexperienced designer cannot provide such guidance, CODA takes default decisions and still generates a design.

6.0 Conclusions

Advances in knowledge engineering hold potential for effective automation of software design methods. This paper presented a knowledge-based approach, integrating semantic data modeling with production rules, for automating CODARTS, a software design method for concurrent and real-time systems. The approach leads directly to an automated designer's assistant, CODA, that was applied to generate ten designs for four real-time problems. CODA compares favorably with other approaches for automating software design methods.

7.0 References

[1] H. Gomaa, Software Design Methods for Concurrent and Real-Time Systems, Addision-Wesley Publishing Company, Reading Massachusetts, 1993.

[2] P. Freeman, "The Nature of Design", Tutorial on Software Design Techniques, (Freeman and Wasserman, eds.), IEEE Computer Society, April 1980, pp. 46-53.

[3] J. Karimi and B. Konsynski, "An Automated Software Design Assistant", *IEEE Transactions on Sofware Engineering*, February 1988, pp. 194-210.

[4] J. Tsai and J. Ridge, "Intelligent Support for Specifications Transformation", *IEEE Software*, November 1988, pp. 28-35.

[5] E. Yourdon and L. Constantine, Structured Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[6] G. Boloix, P. Sorenson, and J. Tremblay, "Transformations using a meta-system approach to software development", *Software Engineering Journal*, November 1992, pp. 425-437.

[7] K. Lor and D. Berry, "Automatic Synthesis of SARA Design Models From Systems Requirements", *IEEE Transactions on Software Engineering*, December 1991, pp. 1229-1240.

[8] D. Parnas, "On Criteria To Be Used in Decomposing Systems into Modules", *Communications of the ACM*, December 1972.

[9] P. Ward and S. Mellor, Structured Development for Real-time Systems, Four Volumes, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[10] D. Hatley and I. Pirbhai, Strategies for Real-Time System Specification, Dorset House, 1987.

[11] K. Mills, Automated Generation Of Concurrent Designs For Real-Time Software, Ph.D. Dissertation, George Mason University, 1996.

[12] National Aeronautics and Space Administration, Software Technology Branch, CLIPS Reference Manual, Three Volumes, CLIPS Version 6.0, June 2, 1993.